

Algorithmes Gloutons

April 13, 2022

1 Principe des algorithmes gloutons

On va considérer un problème pour lequel on est capable de construire une solution de manière incrémentale et d'estimer la qualité des solutions partielles. Pour se fixer les idées, imaginons un labyrinthe dans lequel on avance vers la sortie et où, à chaque croisement, c'est-à-dire au moment de choisir la suite de notre chemin solution, on peut estimer la direction qui *à l'air* de nous rapprocher de la sortie, par exemple en prenant la distance *à vol d'oiseau*. Une telle stratégie est pleine de bon sens, mais peut très bien ne pas conduire à la sortie. En effet, rien n'empêche d'avoir une impasse juste à proximité de la sortie qui serait, elle, accessible en effectuant un grand détour.

Une telle stratégie consistant, face à un choix, à prendre celui qui maximise le gain **à court terme** s'appelle une stratégie gloutonne (*greedy* en anglais). De même, un algorithme résolvant un problème en appliquant une stratégie gloutonne à chaque étape est appelé un **algorithme glouton**.

2 Algorithme glouton pour le rendu de monnaie

On va considérer le problème suivant : étant donné n types de pièces de valeurs faciales respectives $p = (p_0, \dots, p_{n-1})$ et un montant x , on appelle *rendu de monnaie pour x* un n -uplet $c = (c_0, \dots, c_{n-1})$ tel que $S(p, c) = \sum_{i=0}^{n-1} c_i p_i = x$. On cherche un rendu de monnaie minimisant le nombre $N(c) = \sum_{i=0}^{n-1} c_i$ de pièces utilisées.

Par exemple, avec les valeurs $p = (1, 2, 5, 10)$ on peut obtenir $28 = 2 \times 10 + 1 \times 5 + 1 \times 2 + 1 \times 1$ avec le quadruplet $c = (1, 1, 1, 2)$ tel que $N(c) = 5$. On aurait aussi pu utiliser $c' = (28, 0, 0, 0)$ tel que $N(c') = 28$ ou encore $c'' = (1, 1, 5, 0)$ avec $N(c'') = 7$. Il est assez facile de se convaincre que c réalise le rendu de monnaie optimale.

Pour représenter ce problème, on va considérer un tableau $p = [1, 2, 5, 10]$ pour les valeurs et on cherchera à obtenir un tableau $c = [1, 1, 1, 2]$ pour le rendu.

```
[33]: def N(c):  
    # Renvoie N(c) où c est une distribution de pièces  
    # BEGIN SOLUTION  
    s = 0  
    for x in c:  
        s = s + x  
    return s  
    # END SOLUTION
```

```
[6]: from civ import *
init()
test(N, [], 0)
test(N, [1,1,1,2], 5)
test(N, [28,0,0,0], 28)
valide()
```

OK!: $N([])$ vaut 0
 OK!: $N([1, 1, 1, 2])$ vaut 5
 OK!: $N([28, 0, 0, 0])$ vaut 28
 Tous les tests sont valides

```
[7]: def S(p,c):
    # Etant donnés des valeurs faciales p et une distribution c
    # calcule S(p,c) le montant rendu
    # Comme on a besoin de parcourir p et c en même temps on supposera
    # que p et c sont de même longueur sans avoir besoin de s'en assurer
    # BEGIN SOLUTION
    s = 0
    for i in range(len(p)):
        s = s + p[i] * c[i]
    return s
    # END SOLUTION
```

```
[9]: from civ import *
init()
test(S, [], [], 0)
test(S, [1,2,5,10], [1,1,1,2], 28)
test(S, [1,2,5,10], [28,0,0,0], 28)
valide()
```

OK!: $S([], [])$ vaut 0
 OK!: $S([1, 2, 5, 10], [1, 1, 1, 2])$ vaut 28
 OK!: $S([1, 2, 5, 10], [28, 0, 0, 0])$ vaut 28
 Tous les tests sont valides

Étant donné un montant x est des pièces $p = (p_0, \dots, p_{n-1})$, une stratégie gloutonne consiste à chercher la plus grande pièce pouvant être utilisée p_i et à continuer avec $x - p_i$.

```
[1]: def choix_glouton(x, P):
    # Renvoie l'indice i tel que P[i] est la plus grande valeur dans P
    # qui soit <= x. On supposera P trié par ordre croissant et qu'une telle
    # valeur existe.
    # BEGIN SOLUTION
    i = len(P)-1
    while P[i] > x:
        i -= 1
    return i
```

```
# END SOLUTION
```

```
[3]: from civ import *
init()
test(choix_glouton, 1, [1,2,5,10], 0)
test(choix_glouton, 3, [1,2,5,10], 1)
test(choix_glouton, 6, [1,2,5,10], 2)
test(choix_glouton, 11, [1,2,5,10], 3)
valide()
```

OK!: choix_glouton(1, [1, 2, 5, 10]) vaut 0
OK!: choix_glouton(3, [1, 2, 5, 10]) vaut 1
OK!: choix_glouton(6, [1, 2, 5, 10]) vaut 2
OK!: choix_glouton(11, [1, 2, 5, 10]) vaut 3
Tous les tests sont valides

```
[4]: def rendu_vide(P):
    # renvoie un rendu vide, c'est à dire un tableau
    # de même taille que P mais rempli de 0
    # BEGIN SOLUTION
    return [ 0 ] * len(P)
    # END SOLUTION
```

```
[ ]: from civ import *
init()
test(rendu_vide, [], [])
test(rendu_vide, [1], [0])
test(rendu_vide, [1,2], [0,0])
test(rendu_vide, [1,1,1,1,1], [0,0,0,0,0])
valide()
```

Pour résoudre ce problème, on va utiliser un algorithme récursif définissant une fonction `rendu_glouton(x,P)` permettant de renvoyer le rendu pour x avec les pièces P .

- `rendu_glouton(0, P)` va renvoyer un rendu vide
- `rendu_glouton(x, P)` va calculer le choix glouton p_i , effectuer le rendu pour $x - p_i$ puis va rajouter p_i dans le rendu.

```
[6]: def rendu_glouton(x, P):
    # BEGIN SOLUTION
    if x == 0:
        return rendu_vide(P)
    i = choix_glouton(x, P)
    r = rendu_glouton(x-P[i], P)
    r[i] += 1
    return r
    # END SOLUTION
```

```
[9]: from civ import *
init()
test(rendu_glouton, 0, [1,2,5,10], [0,0,0,0])
test(rendu_glouton, 1, [1,2,5,10], [1,0,0,0])
test(rendu_glouton, 2, [1,2,5,10], [0,1,0,0])
test(rendu_glouton, 28, [1,2,5,10], [1,1,1,2])
test(rendu_glouton, 14, [1,2,5,7,10], [0,2,0,0,1])
valide()
```

OK!: rendu_glouton(0, [1, 2, 5, 10]) vaut [0, 0, 0, 0]
 OK!: rendu_glouton(1, [1, 2, 5, 10]) vaut [1, 0, 0, 0]
 OK!: rendu_glouton(2, [1, 2, 5, 10]) vaut [0, 1, 0, 0]
 OK!: rendu_glouton(28, [1, 2, 5, 10]) vaut [1, 1, 1, 2]
 OK!: rendu_glouton(14, [1, 2, 5, 7, 10]) vaut [0, 2, 0, 0, 1]
 Tous les tests sont valides

Pour le dernier exemple, on voit qu'avec $P = (1, 2, 5, 7, 10)$ et $x = 14$, on a effectué le rendu $2 \times 10 + 2 \times 4$ alors qu'on aurait pu faire 2×7 . L'algorithme glouton n'est donc pas optimal. On va étudier ici deux stratégies pour obtenir le choix optimal :

- énumérer tous les rendus et trouver le meilleur
- réfléchir (*hint* c'est souvent une bonne idée)

3 Enumération des rendus

Pour énumérer tous les rendus possibles, on va adapter l'algorithme récursif précédent pour en déduire une fonction `enumere_rendus` telle que `enumere_rendus(x,P)` va renvoyer la liste de tous les rendus de monnaie pour x par rapport aux pièces P .

Pour cela on va procéder ainsi :

- si $x = 0$, il n'existe qu'un rendu possible, le rendu vide, on va donc renvoyer [[0, ..., 0]]
- sinon, on calcule pour chaque pièce $p_i \leq x$ la liste l_i des rendus pour $x - p_i$, on en déduit la liste l'_i en rajoutant la pièce p_i utilisée et on renvoie la concaténée des l'_i (on les met bout à bout avec + en Python).

On va commencer par écrire une fonction permettant de rajouter une pièce à chaque rendu. En faisant en sorte de créer un nouveau rendu vide à chaque fois qu'on traite $x = 0$, on pourra être sûr que cet ajout ne perturbera pas les autres rendus.

```
[11]: def rajoute_piece(l, i):
    # rajoute la piece p_i à chacun des rendus de la liste l
    # cette fonction ne renvoie rien, elle se contente de modifier en place les
    ↪rendus
    # BEGIN SOLUTION
    for r in l:
        r[i] = r[i] + 1
    # END SOLUTION
```

```
[30]: def enumere_rendus(x,P):
    # on utilisera rendu_vide(P) dans le cas x == 0 pour s'assurer de créer
    # de nouveaux rendus vides qui seront augmentés à chaque ajout de pièce
    # BEGIN SOLUTION
    if x == 0:
        return [ rendu_vide(P) ]
    res = []
    for i in range(len(P)):
        if P[i] <= x:
            l = enumere_rendus(x-P[i], P)
            rajoute_piece(l, i)
            res += l
    return res
    # END SOLUTION
```

Si vous évaluez votre fonction dans la cellule suivante vous pourrez voir qu'elle effectue beaucoup de répétitions. En effet, on compte plusieurs fois les rendus selon l'ordre dans lequel on a utilisé une pièce. Mais le rendu effectué en prenant une pièce de 1 puis une pièce de 2 ou l'inverse est en fait le même rendu de monnaie pour 3.

```
[29]: enumere_rendus(14, [1,2,5,7,10])
```

```
[29]: [[14, 0, 0, 0, 0],
      [12, 1, 0, 0, 0],
      [12, 1, 0, 0, 0],
      [12, 1, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [12, 1, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [9, 0, 1, 0, 0],
      [12, 1, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [8, 3, 0, 0, 0],
      [9, 0, 1, 0, 0],
      [12, 1, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [8, 3, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [8, 3, 0, 0, 0],
      [8, 3, 0, 0, 0],
      [7, 1, 1, 0, 0],
      [9, 0, 1, 0, 0],
```

[7, 1, 1, 0, 0],
 [7, 0, 0, 1, 0],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [9, 0, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 0, 0, 1, 0],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],

[5, 1, 0, 1, 0],
 [9, 0, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 0, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],

[6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [9, 0, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 0, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [4, 0, 0, 0, 1],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],

[10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],

[5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [9, 0, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 0, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [4, 0, 0, 0, 1],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],

[8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],

[4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],

[4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [2, 1, 0, 0, 1],
 [9, 0, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],

[3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [4, 0, 2, 0, 0],
 [2, 1, 2, 0, 0],
 [2, 0, 1, 1, 0],
 [7, 0, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [2, 0, 1, 1, 0],
 [4, 0, 0, 0, 1],
 [2, 1, 0, 0, 1],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],

[6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],

[5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],

[3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [2, 1, 0, 0, 1],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],

[4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [1, 4, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [1, 3, 0, 1, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [1, 3, 0, 1, 0],

[7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [1, 3, 0, 1, 0],
 [2, 1, 0, 0, 1],
 [9, 0, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [4, 0, 2, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [4, 0, 2, 0, 0],
 [2, 1, 2, 0, 0],

[2, 0, 1, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [4, 0, 2, 0, 0],
 [2, 1, 2, 0, 0],
 [2, 1, 2, 0, 0],
 [2, 0, 1, 1, 0],
 [7, 0, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [2, 0, 1, 1, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [1, 3, 0, 1, 0],
 [2, 0, 1, 1, 0],
 [4, 0, 0, 0, 1],
 [2, 1, 0, 0, 1],
 [2, 1, 0, 0, 1],
 [12, 1, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [7, 1, 1, 0, 0],

[10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [7, 1, 1, 0, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],

[6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],

[3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [2, 1, 0, 0, 1],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],

[6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [5, 2, 1, 0, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [8, 3, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],

[4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [3, 3, 1, 0, 0],
 [6, 4, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [4, 5, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [2, 6, 0, 0, 0],
 [1, 4, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [1, 3, 0, 1, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [3, 2, 0, 1, 0],
 [1, 3, 0, 1, 0],
 [7, 1, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 2, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [3, 3, 1, 0, 0],
 [1, 4, 1, 0, 0],
 [2, 1, 2, 0, 0],
 [5, 1, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [3, 2, 0, 1, 0],
 [1, 3, 0, 1, 0],
 [2, 1, 0, 0, 1],
 [10, 2, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],
 [8, 3, 0, 0, 0],

```

[6, 4, 0, 0, 0],
[8, 3, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[5, 2, 1, 0, 0],
[8, 3, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[4, 5, 0, 0, 0],
[5, 2, 1, 0, 0],
[8, 3, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[4, 5, 0, 0, 0],
[6, 4, 0, 0, 0],
[4, 5, 0, 0, 0],
[4, 5, 0, 0, 0],
[3, 3, 1, 0, 0],
[5, 2, 1, 0, 0],
[3, 3, 1, 0, 0],
[3, 2, 0, 1, 0],
[8, 3, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[6, 4, 0, 0, 0],
[4, 5, 0, 0, 0],
[6, 4, 0, 0, 0],
[4, 5, 0, 0, 0],
[4, 5, 0, 0, 0],
[3, 3, 1, 0, 0],
[6, 4, 0, 0, 0],
[4, 5, 0, 0, 0],
[4, 5, 0, 0, 0],
...]
```

Une solution va consister à les énumérer en s'assurant que si on utilise p_i à un moment, alors ensuite, on n'utilise pas de pièce p_j où $j > p_i$. *Pourquoi est-ce que cela suffit ?*

On rajoute ainsi un nouveau paramètre à la fonction d'énumérations donnant l'indice i maximal des pièces pouvant être utilisées. Pour obtenir le résultat complet, il suffit alors d'appeler `enumere_rendus_lim(x, P, len(P)-1)`.

```

[27]: def enumere_rendus_lim(x,P,j):
        # BEGIN SOLUTION
        if x == 0:
            return [ rendu_vide(P) ]
```

```

res = []
for i in range(j+1):
    if P[i] <= x:
        l = enumere_rendus_lim(x-P[i], P, i)
        rajoute_piece(l, i)
        res += l
return res
# END SOLUTION

```

On peut ainsi vérifier qu'on obtient chaque rendu uniquement ici :

```
[28]: enumere_rendus_lim(14, [1,2,5,7,10], 4)
```

```

[28]: [[14, 0, 0, 0, 0],
      [12, 1, 0, 0, 0],
      [10, 2, 0, 0, 0],
      [8, 3, 0, 0, 0],
      [6, 4, 0, 0, 0],
      [4, 5, 0, 0, 0],
      [2, 6, 0, 0, 0],
      [0, 7, 0, 0, 0],
      [9, 0, 1, 0, 0],
      [7, 1, 1, 0, 0],
      [5, 2, 1, 0, 0],
      [3, 3, 1, 0, 0],
      [1, 4, 1, 0, 0],
      [4, 0, 2, 0, 0],
      [2, 1, 2, 0, 0],
      [0, 2, 2, 0, 0],
      [7, 0, 0, 1, 0],
      [5, 1, 0, 1, 0],
      [3, 2, 0, 1, 0],
      [1, 3, 0, 1, 0],
      [2, 0, 1, 1, 0],
      [0, 1, 1, 1, 0],
      [0, 0, 0, 2, 0],
      [4, 0, 0, 0, 1],
      [2, 1, 0, 0, 1],
      [0, 2, 0, 0, 1]]

```

```

[39]: def rendu_optimal(x, P):
      # renvoie un rendu optimal, c'est à dire de plus petite valeur N
      # BEGIN SOLUTION
      l = enumere_rendus_lim(x, P, len(P)-1)
      j = 0
      m = N(l[0])
      for i in range(1, len(l)):

```

```

        if N(l[i]) < m:
            m = N(l[i])
            j = i
    return l[j]
# END SOLUTION

```

```

[40]: from civ import *
      init()
      test(rendu_optimal, 14, [1,2,5,7,10], [0,0,0,2,0])
      test(rendu_optimal, 28, [1,2,5,10], [1,1,1,2])
      valide()

```

OK!: rendu_optimal(14, [1, 2, 5, 7, 10]) vaut [0, 0, 0, 2, 0]

OK!: rendu_optimal(28, [1, 2, 5, 10]) vaut [1, 1, 1, 2]

Tous les tests sont valides

4 Vers une meilleure solution

On va reprendre l'énumération précédente mais, au lieu d'accumuler les rendus, on va calculer la pièce permettant de faire le choix optimal. En effet, si la plus grande pièce utilisée dans un rendu optimal $(c_0, \dots, c_{n-1})$ de x est p_i alors $(c_0, \dots, c_{i-1}, c_i - 1, 0, \dots, 0)$ est un rendu optimal de $x - p_i$.

Si on note $Opt(x, P)$ la valeur minimale $N(c)$ pour un rendu de x avec P , on a $Opt(0, P) = 0$ et pour $x > 0$:

$$Opt(x, P) = 1 + \min_{p \in P, p \leq x} Opt(x - p, P)$$

```

[46]: def opt(x, P):
      # Écrire la fonction récursive Opt(x,P)
      # BEGIN SOLUTION
      if x == 0:
          return 0
      m = opt(x-P[0], P)
      for p in P:
          if p <= x and opt(x-p, P) < m:
              m = opt(x-p, P)
      return 1 + m
      # END SOLUTION

```

```

[49]: from civ import *
      init()
      test(opt, 14, [1,2,5,7,10], 2)
      test(opt, 11, [1,2,5,10], 2)
      valide()

```

OK!: `opt(14, [1, 2, 5, 7, 10])` vaut 2
 OK!: `opt(11, [1, 2, 5, 10])` vaut 2
 Tous les tests sont valides

Si on regarde ce qui se passe, on se rend compte que l'appel à `opt(14, [1,2,5,7,10])` effectue 298079 appels récursifs ! Pourtant, comme x ne fait que décroître, il n'y a que 15 valeurs possibles. Cela signifie que la fonction précédente s'appelle continuellement.

Une manière de résoudre ce problème est d'utiliser un dictionnaire qui va nous servir de cache comme on l'a fait pour Fibonacci dans le TP6. C'est-à-dire que le cache est vide au départ, quand on cherche la valeur pour x , si x est dans le cache, on renvoie la valeur associée, sinon on la calcule et on rajoute une nouvelle correspondance.

```
[59]: def opt_cache(x, P, cache):
      # Écrire la fonction récursive Opt en utilisant le cache pour ne pas
      # calculer deux fois une valeur
      # BEGIN SOLUTION
      if x in cache: return cache[x]
      if x == 0:
          v = 0
      else:
          m = opt_cache(x-P[0], P, cache)
          for p in P:
              if p <= x and opt_cache(x-p, P, cache) < m:
                  m = opt_cache(x-p, P, cache)
          v = 1 + m
      cache[x] = v
      return v
      # END SOLUTION
```

```
[61]: from civ import *
      init()
      test(opt_cache, 14, [1,2,5,7,10], {}, 2)
      test(opt_cache, 11, [1,2,5,10], {}, 2)
      test(opt_cache, 28, [1,2,5,10], {}, 5)
      valide()
```

OK!: `opt_cache(14, [1, 2, 5, 7, 10], {})` vaut 2
 OK!: `opt_cache(11, [1, 2, 5, 10], {})` vaut 2
 OK!: `opt_cache(28, [1, 2, 5, 10], {})` vaut 5
 Tous les tests sont valides

Comme la pièce $1 \in P$, on va nécessairement avoir toutes les valeurs de 0 à x dans le cache. On peut ainsi déduire du cache après calcul de $Opt(x, P)$ un tableau $TOpt$ tel que $TOpt[y] = Opt(y, P)$ pour $0 \leq y \leq x$.

```
[62]: def topt(x, P):
      # BEGIN SOLUTION
      cache = {}
```

```

opt_cache(x, P, cache)
t = []
for y in range(x+1):
    t.append(cache[y])
return t
# END SOLUTION

```

```

[65]: from civ import *
init()
test(topt, 14, [1,2,5,7,10], [0, 1, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 3, 2])
test(topt, 10, [1,2,5,10], [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 1])
valide()

```

OK!: topt(14, [1, 2, 5, 7, 10]) vaut [0, 1, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 3, 2]

OK!: topt(10, [1, 2, 5, 10]) vaut [0, 1, 1, 2, 2, 1, 2, 2, 3, 3, 1]

Tous les tests sont valides

5 Reconstruction du rendu optimal

On peut être un peu déçu de la conclusion de la partie précédente : on a obtenu la valeur optimale $N(c)$ mais on ne connaît pas le rendu c qui la réalise. En fait, connaissant $Opt(x, P)$, il est possible de reconstruire un rendu optimal explicitement.

On va noter $ROpt(x, P)$ un rendu optimal et on peut l'obtenir par récurrence ainsi :

- $ROpt(0, P) = [0, \dots, 0]$
- si $x > 0$ et que p est la valeur réalisant $\min_{p \in P, p \leq x} Opt(x - p, P)$, on calcule $(c_0, \dots, c_{n-1}) = ROpt(x - p, P)$ et il suffit de rajouter la pièce p pour obtenir l'optimal.

Afin de simplifier les calculs, on va passer le tableau $TOpt$ en paramètre pour ne pas avoir à recalculer Opt .

```

[70]: def ropt(x, P, T):
    # BEGIN SOLUTION
    if x == 0: return rendu_vide(P)
    m = 0
    for i in range(len(P)):
        if P[i] <= x and T[x-P[i]] < T[x-P[m]]:
            m = i
    r = ropt(x-P[m], P, T)
    r[m] += 1
    return r
    # END SOLUTION

```

```

[72]: from civ import *
init()
T = topt(14, [1,2,5,7,10])

```

```
test(ropt, 14, [1,2,5,7,10], T, [0,0,0,2,0])  
valide()
```

OK!: ropt(14, [1, 2, 5, 7, 10], [0, 1, 1, 2, 2, 1, 2, 1, 2, 2, 1, 2, 2, 3, 2])
vaut [0, 0, 0, 2, 0]

Tous les tests sont valides